From Research to Operations: Integrating Components of an Advanced Diagnostic System with an Aspect-Oriented Framework[1,2]

Daryl P. Fletcher
Science Applications International Corporation (SAIC)
NASA Ames Research Center
Moffet Field, CA. 94035
650-604-0159
dpfletcher@mail.arc.nasa.gov

Richard L. Alena
NASA Ames Research Center
Moffet Field, CA. 94035
650-604-0262
Richard.L.Alena@nasa.gov

Faisal Akkawi
Illinois Institute of Technology
Chicago, IL. 60616
312-567-5122
akkawif@iit.edu

Daniel P. Duncavage
International Space Station Program
NASA Johnson Space Center
Houston, TX. 77058
281-792-5478
Daniel.P.Duncavage@nasa.gov

Abstract—Bringing technology from the research world into an operational environment poses many challenges. Typically, software systems having their origins in low Technical Readiness Level research projects have few, if any, formal requirements associated with them. This paucity of formal requirements coupled with the challenges associated with coordinating multiple, distributed research-oriented software projects makes it difficult to design and build software systems that will ultimately be useful in an operational environment.

Targeted for current and next-generation space vehicles, the diagnostic applications that compose the Advanced Diagnostic System (ADS) under development in our lab at NASA-Ames Research Center are realizations of research projects associated with multiple organizations and generally are not designed according to stringent requirements nor with integration into the ADS environment in mind. The core functionality of a Diagnostic Client Application, usually having its basis in artificial intelligence research, is the primary (and perhaps sole) consideration of the application developer. Research funds generally are not available for implementing aspects such as logging and security, both of which are critical in aerospace diagnostic systems such as the ADS. In order to leverage funding sources and integrate these separate research projects into a coherent whole suitable for deployment in an operational environment, it is important for the systems integrator to be able to easily (affordably) weave these aspects into the software system. Furthermore, Diagnostic Client Applications produce knowledge products, such as subsystem state estimates, that other applications within the ADS universe can use to augment their awareness of the larger system with the goal of generally increasing the effectiveness of the ADS. Taking a holistic approach to knowledge sharing in a software system developed by multiple, loosely-coupled research projects poses a challenge to the systems integrator that goes beyond fundamental inter-process communication issues. However, the challenge can be met and the goal of a more effective ADS can be achieved through the use of emerging data representation technologies.

Aspect-Oriented Programming (AOP) is a new software development methodology that complements Object-Oriented Programming and addresses the complexity of software systems by achieving a separation of functional and interaction components (aspects). Aspects such as logging and security are defined as properties that cut across groups of functional components (diagnostic applications). Aspects can be thought about and analyzed separately from each other and from the core functionality of the software system. AOP provides the modular separation of crosscutting concerns, where the aspect code is scattered/tangled throughout the software system. An AOP Framework takes advantage of what AOP provides and enables us to build software systems that can be extended and adapted during runtime.

In this paper we present an AOP Framework for integrating software components into an Advanced (artificial intelligence-based) Diagnostic System and introduce a basic ontology for sharing knowledge between a community of diagnostic applications (agents). The ontology and AOP Framework can be applied to the development of diagnostic and prognostic decision support systems for current as well as next-generation

[1] U.S. Government work not protected by U.S. copyright.
[2] IEEEAC paper #1333, Version 1, Updated August 27, 2003

space vehicles such as the International Space Station and Orbital Space Plane.

TABLE OF CONTENTS

1. INTRODUCTION

Software development and maintenance is a major cost center of current projects in the NASA International Space Station (ISS) Program, yet the final requirements for software are never complete before the majority of the software has been written. This situation is the result of a number of driving forces and is likely to become even more acute in next generation systems. The nature of NASA projects drives us into new realms of design and therefore unforeseeable problems, which result in changes to software. In addition to requirements that change as a result of technological unknowns, the evolution of security policies and application traceability/transparency requirements impact the software development process. These changes to requirements, as well as typical defect corrections, drive costs both during the design/build/test phase and in the sustaining engineering phase.

To address the issues of vague and/or changing requirements in a complex software development project such as the Advanced Diagnostic System (ADS), we use Aspect-Oriented Programming (AOP) [1, 2, 3, 4, 5, 6, 7] methodology to separate a program's functional components from the interaction components (aspects). Aspects are defined as properties that cut across groups of functional components. While these aspects can be thought about and analyzed relatively separately from the basic functionality, at the implementation level they must be combined together. Programming them manually into the system's functionality using current component-oriented languages results in aspects being spread throughout the code. This code tangling makes the source code difficult to develop, understand and evolve because it destroys modularity and reduces software quality [6]. In this paper we show how to deploy aspect-oriented technology, which provides an architectural support for the design and development of intelligent concurrent systems. We show how the aspect code can be isolated from the functional components that otherwise would be intermingled with the code of the functional components. Isolating the functional components from the non-functional components, the aspect code, has many attractive benefits: first and foremost it promotes reusability for the functional classes and the aspect classes. It also simplifies the design of complex systems, since the interaction code is separated from the functional code. Using the ADS under development at NASA-Ames Research Center as an example of an intelligent systems software development project, we will show how using an Aspect-Oriented Framework can facilitate building reconfigurable intelligent systems, lower software development and maintenance costs and improve overall software quality.

In addition to the apparent advantages of reconfigurability and isolation of core functionality from other utility-type aspects, intelligent systems naturally benefit from the sharing of knowledge throughout the system, and an agent's usefulness to the community can be enhanced through its understanding of the environment in which is exists. There are several mechanisms through which knowledge sharing and reuse can be accomplished including simple lookup tables and distributed object technologies such as XML-RPC and CORBA. However, recent work on Semantic Web[3] technologies has produced a rich set of tools and concepts specifically designed for the representation of data in web-based distributed systems such as the ADS. The OWL Web Ontology Language[4] (a W3C Candidate Recommendation) has been developed specifically to describe and represent a body of knowledge in a computer-usable format. While this is still nascent technology, the potential benefits of using an ontology based on emerging standards exceeds the potential risks.

In section 2 we provide an overview of the Dynamic Weaver Framework architecture in sufficient detail to give the reader a basic understanding of its usage. A more in-depth treatment can be found in the DWF documentation that will be made available for download in early 2004 along with the entire DWF implementation. Section 3 provides an overview of the ADS architecture and presents three Diagnostic Client Applications, still in early stages of development, that are prime candidates for integration into the ISS ADS using the DWF. In section 4 we discuss the problem of information sharing in the ADS and present a basic ontology for knowledge sharing and reuse based on the OWL Web Ontology Language. We conclude by discussing the benefits derived from using the DWF and ADS ontology as tools for moving research-oriented software projects toward operational deployment.

2. THE DYNAMIC WEAVER FRAMEWORK

The Dynamic Weaver Framework (DWF) is an aspect oriented language independent framework. It achieves the separation of concerns by separating the properties of the

[3] W3C Semantic Web

[4] Frequently Asked Questions on W3C's Web Ontology Language (OWL)

system such as logging, security, scheduling, etc., from the functionality of the system then it weaves them together at run time to achieve the overall application system. The DWF employs Java reflection to use the dynamic proxy[4] in order to achieve dynamic adaptability at run time. In this framework, aspects can be added and removed from the system during run time without the need to take the system down to recompile the code. The framework has the ability to attach/detach any aspect to/from the running system, establish communication between any two modules in the system, or redirect communication from one component to the other.

The DWF enables applications to adapt to environmental changes at run time, because components and aspects are independent of each other and they are woven into the system at run time. The components and the aspects in the DWF must have a predefined interface, but the users are free to change the class implementation at run time. The DWF provides us with the capability of adding and removing aspects as well as "point cuts" during runtime. This capability will enable us to support reconfigurability and dynamically adapt to changes in the deployment environment with minimal impact on the running system.

The Dynamic Weaver Framework has the following advantages:

1. We achieve a high level of abstraction since the designer makes the programmer's job easier by reasoning about individual concerns in isolation from each other.

2. Concern reuse. Separation of crosscutting concerns provides the software with a loose coupling between the different concerns, achieving the usability of a single concern.

3. No restrictions are imposed by the software application specification. In aspect-oriented software development, software applications must define when they are going to be adapted at run time by specifying the "join points". In the DWF, the software application is adaptable at run time and its structure can be inspected and dynamically customized, obviating the need to specify what might be adapted and at what time.

4. The DWF is a language independent framework. The system or the software application may be programmed using any language.

5. Application concerns are defined at design level and not at the language or the programming language level, which provides a loose coupling between the design and the implementation, making an aspect more reusable.

6. The DWF achieves a full separation between the functional code and the reusable aspects, which avoids the tangling of application source code, achieving ease of maintenance and adaptation of applications to a new aspect.

In general, the aspect-oriented paradigm has the following elements:

- **Aspect**: The modular representation for a cross-cutting concern. A concern may cross-cut one or more components; security and logging are examples of cross-cutting concerns.
- **Core functional component**: A set of software modules that together contribute to the basic identity of the larger system.
- **Weaver**: The engine that weaves aspects along with their respective core functional components.
- **Join point**: Determines the granularity of the weaving process. In the DWF it is at the method call level.
- **Point cut**: An aspect may have different implementations for different methods; a point-cut represents the specific aspect implementation that will be associated with a specific method(s) of the core functional components. For example, different security policies may be applied to different methods defined in the same component.
- **Advice**: The actual code that will be executed when the control flow reaches the join-points.

Dynamic Weaver Framework Architecture

In this section we provide an overview of the main DWF components. A complete description of the DWF components can be found in the DWF documentation[5].

Aspect Weaver—The AspectWeaver class takes advantage of the dynamic proxy [4] capability in Java 1.3. The framework structure is depicted in the class diagram in Figure 1. Each class uses a dynamic proxy class, which represents the aspect weaver class. A software system has a number of aspects, and each aspect has a number of point cuts. Each point cut has an advice class containing the two methods beforeAdvice() and afterAdvice() that will be executed when control reaches the join points. The semantics of aspect, point cut, and advice are similar to the ones cited in AspectJ[6].

The AspectWeaver weaves classes and their perspective aspects at runtime. The AspectWeaver intercepts messages coming to the component and redirects them to the AspectRepository. The AspectRepository stores information about the aspect(s) (e.g. scheduling, synchronization, security...) and the order in which they have to be executed. The DWF has a loose coupling between the component and the aspects, because the component and the aspects do not have direct references between them.

The AspectWeaver interacts with the clients and does the actual weaving of aspects while the application is running. All communication between the functional

[5] The DWF and documentation will be available for download in early 2004.
[6] Eclipse Projects-AspectJ

components and the aspects of the system is accomplished through the AspectWeaver class.

As mentioned above the AspectWeaver is a dynamic proxy, which directly interacts with the clients and does the actual weaving of aspects while the application is running. Whenever a client calls an aspect method, the AspectWeaver executes the invoke method in the AspectWeaver class. The invoke method in turn executes the AspectRepository's beforeAdvice() method. If the call is successful, the actual operation will be executed. When the task is finished, the AspectWeaver will invoke the afterAdvice() method in the AspectRespository.

In summary, the AspectWeaver class plays two roles in the DWF. First, it intercepts every method called by the system and redirects communication to the AspectRepository by invoking the beforeAdvice() method of the AspectRespository. Second, The AspectWeaver class is responsible for weaving aspects into core functional components at run time.

Aspect Repository—As mentioned in the AspectWeaver class, every method call to the component is intercepted by the AspectWeaver. The AspectWeaver then delegates responsibility to the AspectRepository to evaluate a set of conditions by invoking its beforeAdvice() method. The AspectRepository then evaluates all required aspects of the calling method. Upon successful return of the beforeAdvice() method, a value of RESUME will be returned to the AspectWeaver in which the AspectWeaver then invokes the method on the core functional component itself. Upon the completion of the execution of the method the AspectWeaver will invoke the afterAdvice() method in the AspectRespository.

Aspect Table—The AspectTable is implemented using a hash and resides in the AspectRespository class. It contains all aspects that have been registered in the framework and is used only in the AspectRepository. Initially the AspectTable is empty and is subsequently loaded with an aspect by calling the addAspect() method in the AspectRespository.

The addAspect() method will insert any aspect name that you provide to the system into the AspectTable along with an index and an object that contains the actual aspect. Similarly, removing an aspect from the AspectTable can be done by calling the removeAspect() method in the AspectRepository class.

Aspects—An AspectTable can contain multiple Aspect objects and each Aspect can contain multiple Pointcuts. We represent an aspect object as a hash table that exists in the AspectTable. Every time an aspect is added to the AspectTable a new hash table is instantiated and inserted into a new row in the AspectTable inside the AspectRepository. Each aspect can contain multiple Pointcuts and every Pointcut may contain one or more join points.

Point cuts—An Aspect can contain multiple Pointcut objects and each Pointcut can contain multiple Advices. The method addAdvice() is used for adding a new advice. The removeAdvice() method removes an advice from a given Pointcut. The beforeAdvice() and afterAdvice() methods are invoked by the corresponding Aspect's methods of the same name.

Advices—The actual behavior of each aspect is provided by an object whose interface is defined by AdviceIF. They will be woven during runtime by the dynamic proxy, i.e., AspectWeaver. The beforeAdvice() method returns one of the integer constants defined in the AspectRepository; RESUME, BLOCK, and ABORT.

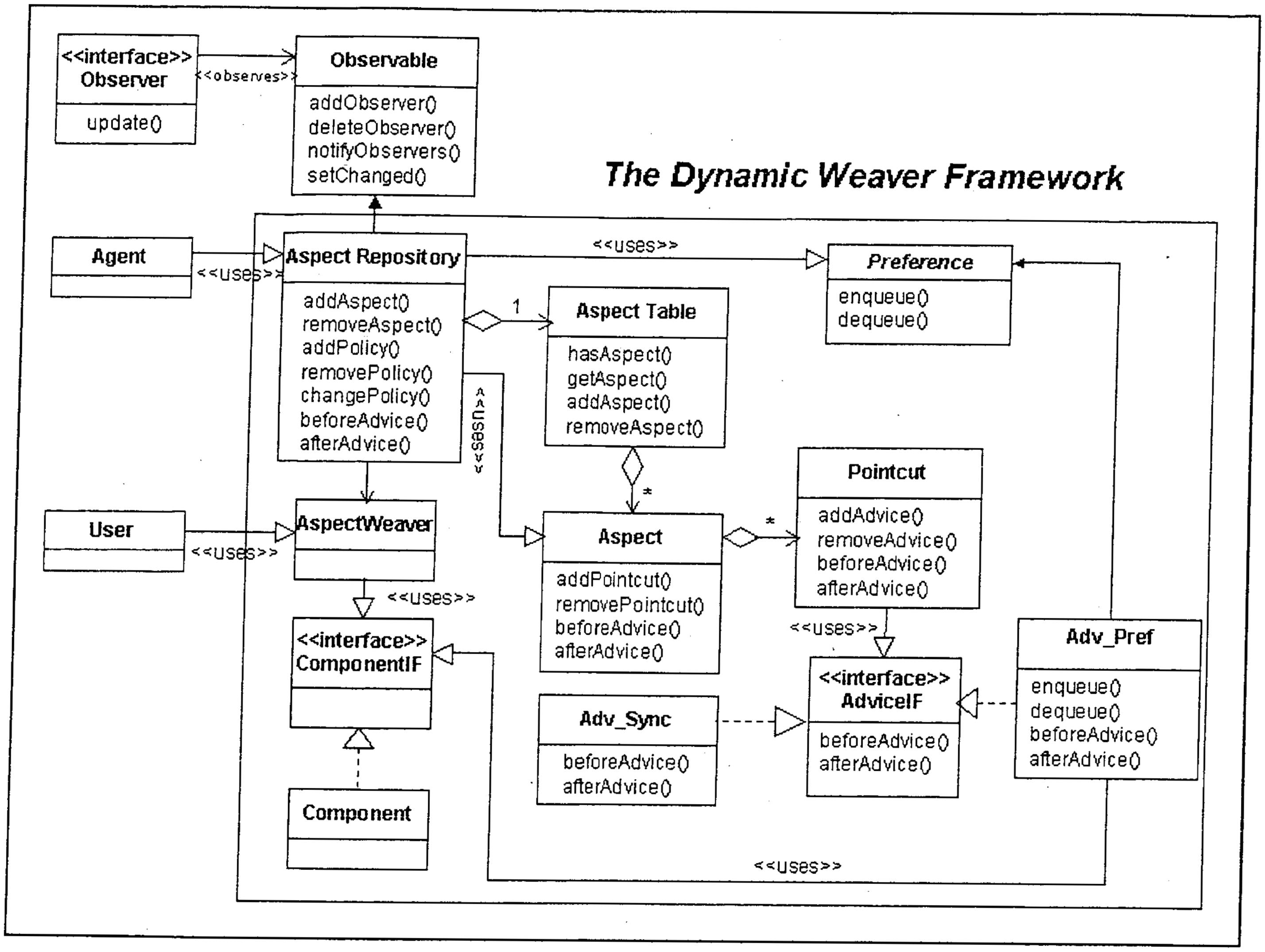

Figure 1: Dynamic Weaver Framework class diagram

3. ADS Component Integration Using the Dynamic Weaver Framework

The Advanced Diagnostic System (ADS) under development in the Intelligent Mobile Technologies (IMT) Lab at NASA-Ames Research Center [13] is an intelligent decision support system for current and next-generation space vehicles such the International Space Station (ISS) and Orbital Space Plane (OSP). In the following sections we show how ADS development can benefit from using the Dynamic Weaver Framework for integrating components into the ADS ecosystem.

The Diagnostic Client Applications (DCAs) that compose the ADS are realizations of research projects associated with multiple organizations and generally are not designed according to stringent requirements nor with integration into the ADS in mind. The core functionality of a DCA, usually having its basis in artificial intelligence research, is the primary (and perhaps sole) consideration of the DCA developer. Research funds generally are not available for implementing aspects such as logging, debugging or security, all of which are critical in a flight-qualified system such as the ADS. Furthermore, the logging and security requirements are not well understood at design time and are likely to evolve throughout the development and maintenance phases. It is therefore important for the systems integrator to be able to easily (affordably) weave these aspects into the system, even after the system has been deployed on-orbit.

Advanced Diagnostic System Overview

The ISS ADS architecture is summarized in the following figure:

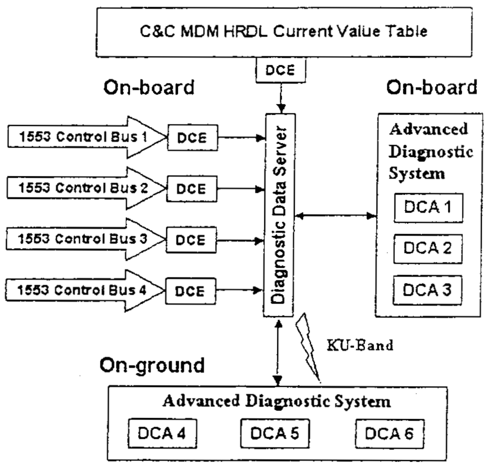

Figure 2: ISS Advanced Diagnostic System Architecture

The Advanced Diagnostic System is composed of a number of Diagnostic Client Applications (DCAs) that reside both on board and on the ground. The DCAs obtain pre-processed avionics data from the on-board Diagnostic Data Server (DDS) using a publish/subscribe architecture. Diagnostic clients are generally artificial intelligence-based applications such as artificial neural networks, Bayesian belief networks, fuzzy cognitive maps and model-based reasoners that integrate data across sub-systems. The collaborative product of the DCAs is the Advanced Diagnostic System, a decision support tool targeted for use by crew, flight controllers and back-room engineering groups on the ground.

The DDS inputs are provided by Data Collection Engines (DCEs) that interact directly with trusted, lower-level avionics components such as MIL-STD 1553 data buses and the Command and Control Multiplexer/Demultiplexer (MDM) Current Value Table, a shared memory construct of the ISS Command and Control Software. Core DDS functions create derived data products, such as Caution and Warning messages, and make them available to DCAs through a publish/subscribe architecture. A DCA can share data products with its peers using the DDS publish/subscribe mechanism.

The MIL-STD 1553 protocol used for the data bus on the ISS is a synchronous, deterministic protocol that uses three data rates to transport parameters throughout the C&C system: 10 Hz, 1 Hz and 0.1 Hz. The DCEs that interact with the MIL-STD 1553 buses must process the raw data within the boundaries of these three data rates and are therefore hard real-time tasks. The pre-processing functions associated with the DDS can buffer data coming from the DCEs and are categorized as soft real-time tasks.

Advanced Caution and Warning Diagnostic Client Application Overview

In this section, we focus on a the Advanced Caution and Warning (ACW) Diagnostic Client Application (DCA), composed of an alarm filtering function, implemented as an Artificial Neural Network (ANN) and an alarm correlation function, implemented as a Bayesian Network (BN)[7].

Fault detection and isolation (FDI) is a function of the Advanced Diagnostic System that uses messages (alarms) created by the Caution and Warning software of the complex system under management (diagnosis). These caution and warning messages are published to a subscriber DCA (the Advanced Caution and Warning filtering function) by the Diagnostic Data Server. In a complex system such as the International Space Station or Orbital Space Plane, a component or subsystem fault event can generate tens, hundreds or even thousands of alarms (an 'alarm storm'), some significant percentage of which may not be related to the root cause of the fault event. Operations personnel, either through training or experience, can visually filter some portion of the unrelated alarms (noise) in order to reduce the sample space of probable root causes (isolation). Time is usually

[7] Also called a *belief network* or *causal network*.

critical during FDI activities and the Advanced Caution and Warning filtering function reduces the noise in the alarm stream, aiming to reduce the time required for FDI. Once the noise has been filtered from the alarm set, alarm correlation can be performed either by a human or automated process. When an automated process such as a Bayesian Network is used for alarm correlation, the ACW filtering function acts as a preprocessor to reduce the size of the input vector, thereby speeding up convergence.

A supervised learning technique is used to train the ANN alarm filter to recognize and filter extraneous alarms based on the current state (mode) of the ISS. For instance, when the ISS mode transitions from "Standard" to "Proximity Operations" in preparation for a planned docking event, certain nuisance alarms are generated that can be safely filtered from the alarm stream. An operational benefit derived from this function is that these nuisance alarms don't have to manually suppressed (and then reactivated) each time one of these extraneous alarm-producing events occur. This manual deactivation/reactivation of alarms is a potential safety risk that we aim to mitigate with the ANN alarm filter.

A Bayesian Network is a *directed acyclic graph*[8] where the nodes represent random variables and the arcs represent the probabilistic relationships between them. The parents of a node *X* are those variables that are judged to be direct causes of *X* or to have direct influence on *X* [15]. A conditional probability is specified for each node (variable) and the graph can be considered as representing the joint probability distribution for all the variables. Sterritt et.al. have shown that a Bayesian Belief Network can be effectively used as part of an intelligent fault management system for telecommunications networks and the ADS BN alarm correlator DCA is largely based on the work presented in that paper [16].

The Java Neural Network Simulator (JavaNNS[9][10]) is the kernel of the ACW filtering function and JavaBayes[11] provides the core inference engine for the alarm correlation function.

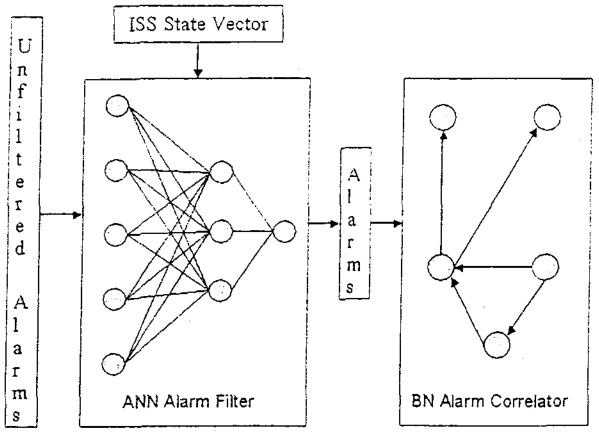

Figure 3: ACW DCA architecture

We integrate the ACW DCA into the ADS using the Dynamic Weaver Framework. The ACW filtering function and the Bayesian Network alarm correlator were designed with minimal security and logging aspects, both of which are important concerns for a system targeted for flight qualification such as the ADS. Using the Dynamic Weaver Framework, previously flight-qualified cross-cutting concerns such as logging and security can be woven into the ACW DCA at runtime thereby eliminating the need to re-qualify these aspects resulting in lower flight qualification costs. As requirements for logging and security inevitably evolve, these cross-cutting concerns can be handled separately from the core functions of the ACW DCA.

A Fuzzy Cognitive Map Toolkit for Decision Support

In addition to the ACW DCA, the current ADS architecture includes a Fuzzy Cognitive Map (FCM) Toolkit that can be used for modeling complex, dynamical systems.

A FCM [8, 9] is a fuzzy, signed directed graph with feedback where the nodes represent *concepts* and a directed edge e_{ij} measures how much concept C_i *causes* Cj. A time varying concept $C_i(t)$ measures the *degree of occurrence* of some event, such as the degree to which a component has failed or the "strength" of subsystem health, and can take on values in the fuzzy interval [0, 1]. The edges e_{ij} take on values in the fuzzy interval [-1, 1] where $e_{ij} = 0$ indicates no causality from Ci to Cj, $e_{ij} > 0$ indicates causal correlation in the same direction (Cj increases as C_i increases or Cj decreases as Ci decreases) and $e_{ij} < 0$ indicates negative causal correlation (Cj decreases as C_i increases or Cj increases as Ci decreases). Operations on the graph can be performed using matrix-vector operations.

Fuzzy Cognitive Maps differ from probabilistic decision support systems such as Bayesian Belief Networks. In a probabilistic context, when a random event occurs (such as the event of "heads" when a coin is tossed), the event occurs completely, i.e. the result of the experiment is

[8] A directed graph where no path starts and ends at the same vertex.

[9] I. Fischer, F. Hennecke, C. Bannes, A.Zell: Java Neural Network Simulator User Manual, Version 1.1, University of Tubingen

[10] JavaNNS is Copyright (c) 1996-2001 JavaNSS Group, Wilhelm-Schickard-Institute for Computer Science (WSI), University of Tubingen, Sand 1, 72076 Tubingen, Germany.

[11] JavaBayes is distributed under the GNU General Public License.

either entirely heads or entirely tails. Comparatively, events in a FCM occur deterministically but to varying degrees, such as "light rain" or "bright sunshine". If a fuzzy event is non-deterministic, we can integrate the two approaches to create a compound statement describing the probability of a fuzzy event, such as "a 20 percent chance of light rain".

A FCM is usually constructed by a knowledge engineer who acquires domain knowledge from systems experts and uses that knowledge to define the concepts, causal directions and fuzzy values of the nodes and edges of the graph. As an example, consider the High Level ISS System Health Monitor DCA shown below implemented as a FCM representing the causal relations of the fuzzy concepts of Command and Data Handling (C&DH) Subsystem health (C1), Electrical Power Subsystem (EPS) health (C2), Thermal Control Subsystem (TCS) health (C3), Channel 2B battery low (C4) and Low Temperature Loop (LTL) heat exchanger can't reject heat (C5):

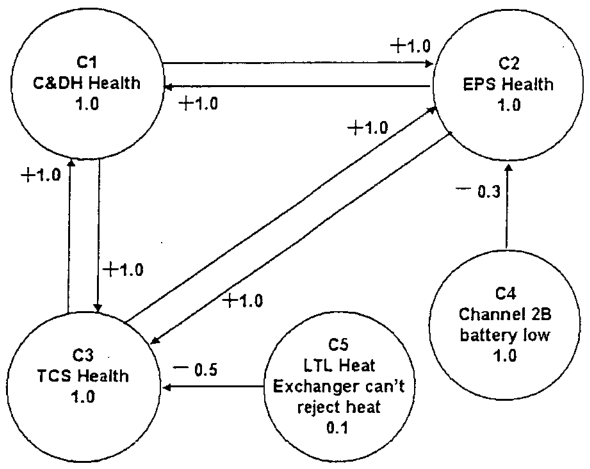

Figure 4: A High-level ISS Systems Health Monitor DCA implemented as a Fuzzy Cognitive Map

Subsytem experts have different opinions regarding the degree to which the health of one subsystem affects the health of others. The knowledge engineer can conduct multiple interviews with subsystem experts and combine each of the resulting FCMs to construct a new FCM that cumulatively embodies the knowledge of each of the experts. A weighting function can be used to give more weight to a FCM constructed from an interview with a more experienced systems engineer and a lesser weight to one constructed on advice from a less experienced systems engineer. The resulting FCM is a linear combination of the separate FCMs:

$$\mathbf{F} = \sum w_i \mathbf{F}_i$$

Equation 1: FCM combination

When the FCMs are combined, a threshold function is used to map the connection values to the interval [-1, 1] and the concept values to the interval [0, 1]. The example FCM can be represented as a concept vector:

$$\mathbf{C}_t(\mathbf{0}) = [1.0,\ 1.0,\ 1.0,\ 1.0,\ 0.1]$$

Figure 5: Initial FCM concept vector

and a connection (edge) matrix **E**:

$$\begin{bmatrix} 0.0 & 1.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -.3 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -.5 & 0.0 & 0.0 \end{bmatrix}$$

Figure 6: FCM connection matrix for the FCM in Figure 4

The entries in the concept vector **C(0)** are initial estimates of the concept values given the conditions that the LTL Heat Exchanger has lost 10% of its heat rejection capacity and the Channel 2B battery is completely discharged. The entries in the connection matrix **E** are the result of applying Equation 1 to a series of connection matrices derived from interviews with multiple subsystem experts.

A FCM is a dynamical system that can simulate the behavior of the process being modeled [14]. Successive matrix-vector multiplications are performed with the output of one operation being used as the input to the next. The FCM simulation will either diverge or converge to a fixed point (a single vector) or limit cycle (repeating pattern of vectors). While holding the connection values fixed and "clamping" (firing) the C4 and C5 concepts to simulate the LTL Heat Exchanger and Channel 2B battery failure modes, the FCM converges to a new set of concept values:

$$\mathbf{C(t)} = [0.83,\ 0.80,\ 0.78,\ 0.73,\ 0.73]$$

Figure 7: FCM concept vector after iteration

These new equilibrium values for concepts C1, C2 and C3 are interpreted as follows: given the causal relations represented by the connection matrix and the initial estimates of subsystem health, the computed health measures for the C&DH, EPS and TCS subsystems are 83%, 80% and 78%, respectively, when failure conditions C4 and C5 occur simultaneously. In another simulation, firing C5 alone yields:

$$\mathbf{C(t)} = [0.84,\ 0.84,\ 0.79,\ 0.0,\ 0.73]$$

Figure 8: FCM concept vector after fault condition C4 is removed

showing that the removal of fault condition C4 increases C&DH health by one percent, EPS health by four percent and TCS health by one percent. Removing fault condition C5 yields similar results.

The FCM Toolkit can be used to build more complex models than the previous example and has a Differential Hebbian Learning (DHL) function that can imply a connection matrix from a time series of concept observations. When the number of concepts being modeled becomes large, the DHL function relieves the user of having to construct a graphical FCM by hand and simply uses recorded observations of each concept individually, "learns" the causal strengths according to changes in causation over time and automatically produces a connection matrix that can be used as a model for the complex system.

The ACW DCA, FCM Toolkit and the Dynamic Weaver Framework

The kernel of the ACW artificial neural network (ANN) filtering function, the Bayesian network (BN) core inference engine of the alarm correlation function and the computational engine of the FCM Toolkit are core functional components of the ISS ADS. The ANN filtering function receives input vectors of alarms from the Diagnostic Data Server (DDS) and the BN correlation function receives its input vectors from the ANN filtering function. The BN then publishes its result set back to DDS so that other ADS DCAs can subscribe to the results of the belief network inference. Similarly, the FCM Toolkit core computational engine produces result vectors from ad-hoc simulations that may be of value to other DCAs in the ADS ecosystem and data products of other DCAs can provide useful knowledge for a given FCM model. A security aspect associated with each core functional component ensures that input data is coming from a trusted source and a logging aspect supports transparency so that the end user can see how a DCA has come about its results.

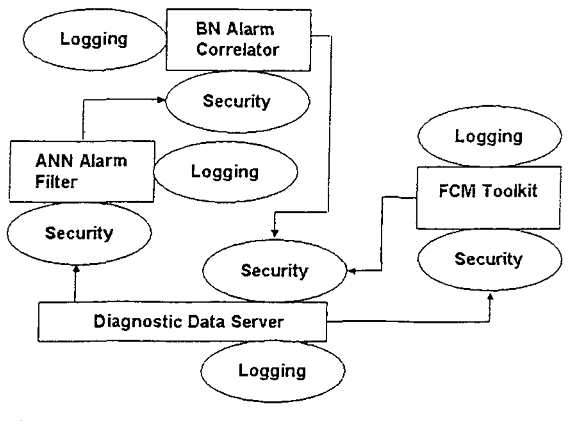

Figure 9: DCA Crosscutting concerns (non-Aspect-Oriented view)

From the above figure, it's clear that the logging and security aspects of the DCAs are scattered throughout the core functional components of the ADS. The functional requirements of these crosscutting concerns are likely to evolve over time, both during the development phase and later when the ADS has been flight qualified and deployed on-orbit. Once a software system has been deployed on orbit, changes to the running system are difficult and expensive. Using the Dynamic Weaver Framework (DWF) we aim to minimize the impact (cost) that a security or logging requirements change will have on the deployed system by achieving a separation between the crosscutting concerns and the core functionality of a DCA. This will allow us to make the code base that is subject to change (and perhaps be re-qualified) as small as possible. Furthermore, using the DWF, the system will be able to adapt to changes dynamically at runtime, eliminating the need to shutdown and/or recompile the operational system to accommodate new logging or security requirements.

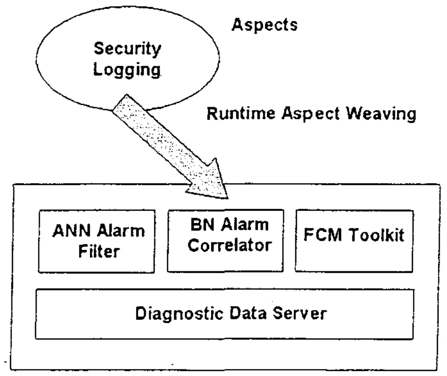

Figure 10: Aspect-Oriented view of the ACW and FCM DCAs

In the DWF, the AspectWeaver class weaves classes and their perspective aspects, such as logging and security, at runtime. In the case of the security aspect, the AspectWeaver intercepts the input vector to the core functional component and redirects it to the AspectRepository. The AspectRepository stores information about the logging and security aspects and the order in which they have to be executed. The DWF provides a loose coupling between the core functional components and the aspects, because the components and the aspects do not have direct references between them. All communication between the core functional components and the aspects of the system is accomplished through the AspectWeaver class.

The AspectWeaver is a dynamic proxy that directly interacts with the clients. In the security aspect, the clients are the methods of the core functional components that get input vectors from their buffers. In the logging aspect, the clients are the methods of the core functional components that write out log messages. Whenever a client calls a logging or security method, the AspectWeaver executes the corresponding invoke() method in the AspectWeaver class. The invoke() method then executes the AspectRepository's beforeAdvice() method. If the call is successful, the actual operation that is a logging or security method will be executed. When the method completes, the AspectWeaver will invoke the afterAdvice() method in the AspectRepository.

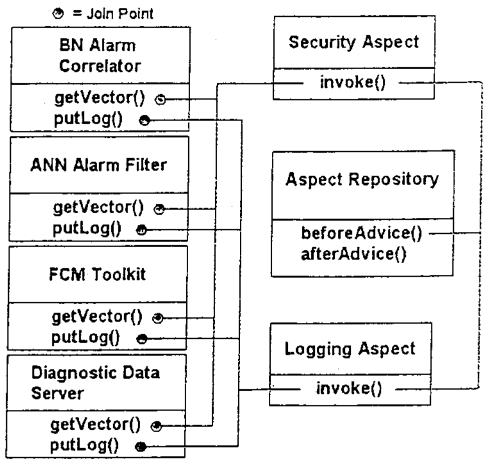

Figure 11: ACW and FCM DCAs in the Dynamic Weaver Framework

By using the Dynamic Weaver Framework to integrate the ACW and FCM DCAs into the ADS environment, we enhance the adaptability of the system and simplify integration of reusable aspects such as logging and security. These benefits provided by the DWF will lessen the impact and costs associated with implementing unforeseeable changes to the theses important crosscutting concerns during the design/build/test and sustaining engineering phases of the software life cycle. The trade-off is a slight performance degradation due to the fact that calls to aspect functions require the additional overhead of making calls through the AspectWeaver rather than directly invoking the aspect function themselves.

We've shown how using the DWF can simplify (and hence lower the cost) of integrating a DCA into the ADS architecture at the cost of a slight performance trade-off, but what happens when disjoint DCAs want to share knowledge about their particular domain? The Framework doesn't address the semantics of knowledge sharing and reuse. In the next section, we introduce a basic ontology that will address these issues within the ADS architecture.

4. Sharing Knowledge in the ADS

Within the ADS, diagnostic client applications produce data products that are useful to other client applications. For instance, the FCM DCA produces an ephemeral concept vector representing the current state of certain components within a given subsystem and the ACW ANN Alarm Filter can use that subsystem state information to augment its knowledge of the system (included in the ISS state vector) in order to more accurately filter extraneous alarms. If the ANN Alarm Filter knows that pertinent state information is available from an agent within the ADS architecture, it doesn't have to go looking for it elsewhere. The BN Alarm Correlator produces a list of root cause candidates for a given subsystem fault that can be used by another DCA concerned with that particular subsystem, e.g. an ad-hoc FCM constructed to analyze a particular C&DH problem. While it is clear that the overall effectiveness of the ADS can be enhanced by the sharing of certain data products between DCAs, it is unclear how that knowledge should be represented and shared within the ADS architecture. Through what mechanism can agents publish their data products and subscribe to the pertinent (and perhaps ephemeral) data products of other agents, dynamically?

An *ontology* defines the vocabulary with which queries and assertions are exchanged among agents (DCAs) [10]. In this section we present a basic ADS ontology that describes *ontological commitments* enabling DCAs to share data products and gain knowledge about the environment in which they exist. The presented ontology is *basic* because it will evolve over time as the concept of ADS expands.

A Basic Ontology for the ADS

We could represent shared knowledge within the ADS using a simple lookup table, but an ontology provides a much richer set of constructs through which we can formally describe the semantics of classes and properties of ADS resources. Furthermore, the ADS ontology can be updated dynamically by the Diagnostic Data Server to reflect the data products currently available at a given point in time.

We use the OWL Web Ontology Language as the basis for our ADS ontology since it is designed for use by applications that need to process the content of information rather than presenting information to humans. OWL builds on web-based information representation languages such as XML, Resource Description Framework (RDF) and RDF Schema (RDF-S) and goes beyond these languages in its ability to represent machine-interpretable content on the Web[12] [11].

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
xmlns:rdf=http://www.w3.org/1999/02/22rdf-syntax-ns#
xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
xmlns:dc=http://purl.org/dc/elements/1.0/
xmlns:owl=http://www.w3.org/2002/07/owl#
xmlns=http://www.w3.org/2002/07/owl#>
<Ontology rdf:about="">
   <dc:title>ADS Ontology</dc:title>
   <dc:creator>IMT Lab</dc:creator>
   <dc:subject>OWL; ADS;</dc:subject>
   <dc:publisher>Daryl Fletcher</dc:publisher>
```

[12] ADS ontology publication is limited to a secured intranet and is not generally available on the Web.

```
<dc:date>2003-09-02</dc:date>
<dc:format>text/xml</dc:format>
<dc:language>en</dc:language>
</Ontology>

<owl:Class rdf:ID="ADS">
<label>Advanced Diagnostic System</label>
</owl:Class>

<owl:Class rdf:ID="DCA">
<label>Diagnostic Client Application</label>
</owl:Class>

<owl:ObjectProperty rdf:ID="Description">
<rdfs:domain><owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#ADS"/>
<owl:Class rdf:about="#DCA"/>
</owl:unionOf>
</owl:Class></rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Owner">
<rdfs:domain><owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#ADS"/>
<owl:Class rdf:about="#DCA"/>
</owl:unionOf>
</owl:Class></rdfs:domain
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Contact">
<rdfs:domain><owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#ADS"/>
<owl:Class rdf:about="#DCA"/>
</owl:unionOf>
</owl:Class></rdfs:domain>
</owl:ObjectProperty>

<owl:Class rdf:ID="ISS_ADS">
<label>ISS Advanced Diagnostic System</label>
<owl:subClassOf rdf:resource="#ADS" />
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#ISS_CDH_DCA"/>
<owl:Class rdf:about="#ISS_ECW_DCA"/>
</owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="ISS_DCA">
<label>ISS Diagnostic Client Application</label>
<owl:subClassOf rdf:resource="#DCA" />
</owl:Class>

<ISS_ADS owl:Class rdf:ID="imt_iss_ads">
<label>IMT Lab ISS ADS</label>
<Description>
An Advanced Diagnostic System for the International
Space Station under development in the Intelligent
Mobile Technologies Lab at NASA-Ames Research
Center
</Description>
<Owner>Dan Duncavage</Owner>
<Contact>daniel.p.duncavage@nasa.gov</Contact>
</ISS_ADS>

<owl:Class rdf:ID="ISS_FCM_DCA">
<owl:subClassOf rdf:resource="#ISS_DCA" />
</owl:Class>

<ISS_FCM_DCA rdf:ID="fcm_toolkit_dca">
<Description>
A core computational engine and set of graphical
tools for modeling complex systems using Fuzzy
Cognitive Maps
</Description>
<Owner>Daryl Fletcher</Owner>
<Contact>dpfletcher@mail.arc.nasa.gov</Contact>
<Publishes>
<ConceptVector>
<ReferenceInformation>
<document>FCM Toolkit
User'sManual</document>
</ReferenceInformation>
<rdfs:comment>
A concept vector consists of a time stamp followed
by a series of comma separated element state
estimations, terminated by a newline.
</rdfs:comment>
<dataProductFormat>
<timestamp rdf:datatype="&xsd;dateTime" \>
<cdh_health rdf:datatype="&xsd;float" \>
<eps_health rdf:datatype="&xsd;float" \>
<tcs_health rdf:datatype="&xsd;float" \>
</dataProductFormat>
<howToSubscribe>
<subscriberInstructions>
Send registered user name and password to the
following host and port along with subscribing
application's IP address and port.
</subscriberInstructions>
<Host>xxx.xxx.xxx.xxx</Host>
<Port>18333</Port>
</howToSubscribe>
</ConceptVector>
</Publishes>
</ISS_FCM_DCA>

<owl:Class rdf:ID="ISS_ECW_DCA">
<owl:subClassOf rdf:resource="#DCA" />
</owl:Class>

<ISS_ECW_DCA rdf:ID="acw_dca">
<Description>
An Advanced Caution and Warning application that
filters alarms using an Artificial Neural Network and
performs fault correlation using a Bayesian Network.
</Description>
<Owner>Daryl Fletcher</Owner>
<Contact>dpfletcher@mail.arc.nasa.gov</Contact>
<Publishes>
<rdfs:comment>
All time stamps in published products are in ISO
8601 Format: yyyy-mm-dd hh:mm:ss.xxx.
</rdfs:comment>
<FilteredAlarms>
```

```
110 <ReferenceInformation>
111 <url>http://www.jsc.nasa.gov/c&w/index.html
112 </url>
114 <document>ISS Familiarization</document>
115 <document>C&DH Training Manual</document>
116 </ReferenceInformation>
117 <rdfs:comment>
118 Fields within alarm event blocks are separated by
commas. Event blocks are delimited by newlines.
119 </rdfs:comment>
120 <dataProductFormat>
121 <eventptr rdf:datatype="&xsd;int" \>
122 <logtime rdf:datatype="&xsd;dateTime" \>
123 <dayofyear rdf:datatype="&xsd;int" \>
124 <event rdf:datatype="&xsd;int" \>
125 <alarmtype rdf:datatype="&xsd;string" \>
126 <ackstate rdf:datatype="&xsd;string" \>
127 <eventstate rdf:datatype="&xsd;string" \>
128 <status rdf:datatype="&xsd;int" \>
129 <annstate rdf:datatype="&xsd;string" \>
130 </dataProductFormat>
131 <howToSubscribe>
132 <subscriberInstructions>
133 Send registered user name and password to the
following host and port along with subscribing
application's IP address and port.
134 </subscriberInstructions>
135 <Host>xxx.xxx.xxx.xxx</Host>
136 <Port>17593</Port>
137 </howToSubscribe>
138 </FilteredAlarms>
139 <RootCauseAnalysis>
140 <ReferenceInformation></ReferenceInformation>
141 <rdfs:comment>
142 A root cause analysis consists of a time stamp
followed by a statement of root cause candidates
terminated by a newline.
143 </rdfs:comment>
144 <dataProductFormat>
145 <timeStamp rdf:datatype="&xsd;dateTime">
146 <rootCauseCandidates rdf:datatype="&xsd;string">
147 </dataProductFormat>
148 <howToSubscribe>
149 <subscriberInstructions>
150 Send registered user name and password to the
following host and port along with subscribing
application's IP address and port.
151 </subscriberInstructions>
152 <Host>xxx.xxx.xxx.xxx</Host>
153 <Port>17594</Port>
154 </howToSubscribe>
155 </RootCauseAnalysis>
156 </ISS_ECW_DCA>
157 </rdf:RDF>
```

Figure 12: The ADS ontology

Lines 1-16—These lines provide the namespace references and form the header of the ontology.

Lines 17-19—The ontology specifies an ADS as a base class that can be subclassed to represent different types of ADSs, e.g. an International Space Station ADS or an Orbital Space Plane ADS.

Lines 20-22—As with the ADS, the ontology specifies a DCA as a base class that can be subclassed to represent different types of DCAs, e.g. an International Space Station DCA or an Orbital Space Plane DCA.

Lines 23-46—These lines assert that the ADS and DCA classes have properties associated with them named *Description*, *Owner* and *Contact*.

Lines 47-54—Define the class ISS_ADS, a subclass of ADS specific to the International Space Station. The class is composed of the union of two DCA subclasses, namely an ISS_CDH_DCA and an ISS_ECW_DCA. Each of these DCA subclasses can have multiple instances representing distinct *individuals*, e.g., there can be several different ISS_CDH_DCAs within the ISS_ADS, each having its own identity.

Lines 55-58—Define the class ISS_DCA, a subclass of DCA specific to the International Space Station. To easily extend the ADS ontology to another domain such as the Orbital Space Plane, we would simply define a class OSP_DCA as another subclass of DCA.

Lines 58-65—Here we introduce an individual instance of an ISS_ADS named *imt_iss_ads*. It has a *Description*, an *Owner* and *Contact* information. We could have another instance of an ISS_ADS developed by another group, say group ABC, and name it *abc_iss_ads*. Then, using the vocabulary established in the ADS ontology, knowledge could be shared between multiple ADSs, similar to the manner in which knowledge is shared among DCAs.

Lines 66-68—Define the class ISS_FCM_DCA, a subclass of ISS_DCA.

Lines 69-95— Here we introduce an individual instance of an ISS_FCM_DCA named *fcm_toolkit_dca*. It has a *Description*, an *Owner* and *Contact* information, as do all DCAs and ADSs. Note the <Publishes> section starting on line 74. Enclosed in this section is specific information about how this DCA goes about sharing its knowledge with the world around it. It has one data product that it wishes to share; a <ConceptVector>. There is a document included in the <ReferenceInformation> section that this DCA believes is relevant to the understanding of its <ConceptVector> data product. Lines 79-80 contain a comment that is a human-readable description of the <ConceptVector> format, while the <dataProductFormat> section in lines 81-86 contains a machine-readable description of the <ConceptVector> format. The <ConceptVector> consists of a time stamp and values for the concepts C&DH health, EPS health and TCS health. The <howToSubscribe> section in lines 87-92 provides information for agents that wish to subscribe to this particular data product[13].

Lines 96-98—Define the class ISS_ECW_DCA, a subclass of ISS_DCA specific to the Emergency, Caution and Warning (ECW) System.

Lines 99-156—Here we introduce an individual instance of an ISS_ECW_DCA named *acw_dca*. This is the Advanced Caution and Warning DCA described in this document. It has a *Description*, an *Owner* and *Contact* information and publishes two data products: <FilteredAlarms> and <RootCauseAnalysis>. Note that the overall ontological structure of *acw_dca* is the same as that of *fcm_toolkit_dca*. The *acw_dca* provides links to information it believes is important for understanding its data products, as well as providing information for potential subscribers. The details of this DCAs data product descriptions are essentially the same as described for the *fcm_toolkit_dca* and are not repeated here.

Line 157—Closes the ADS ontology.

In the ontology presented above, the ADS is a union of classes that can be easily extended to include multiple ADSs and even form a hierarchy of ADSs, much like the "manager-of-managers" hierarchical structure typically found in large-scale Network Management Systems [12] where one ADS could become a DCA of another ADS. The hierarchy of ADSs can evolve along with the evolution of the complex system to which the ADS is applied; smaller ADSs can be developed in parallel with the complex system and then integrated to form a coherent whole while maintaining a consistent representation of ADS concepts. Using the ADS ontology, DCAs within an ADS can dynamically discover and subscribe to ADS resources using standard, web-based technologies. Our ADS ontology is dynamic, scalable, extensible, expressive in its conceptualization of the ADS universe, is easily accessed by distributed agents and is based on emerging standards easily adopted by DCA developers.

5. Conclusion

This paper presents some of the challenges associated with bringing software projects from the research world into an operational environment. While the core functional components of research-oriented software applications can have great utility in an operational setting, these applications often lack aspects important in an operational environment such as logging and security. Furthermore, these stand-alone applications, sometimes developed in isolation from one another, can produce data products useful to other applications in a software ecosystem.

We present the Dynamic Weaver Framework, an Aspect-Oriented framework for dynamically weaving aspects such as logging and security into disparate applications at run-time. Aspect-Oriented methodology isolates code that would otherwise be tangled throughout the software system and separates utility-type aspects from the core functional components, allowing research application developers to focus resources on the research-oriented components of the system. The Framework takes advantage of the benefits of Aspect-Oriented Programming and uses a dynamic proxy for weaving aspects such as logging and security into the software system at run-time, providing the systems integrator with an economical method for bringing lower Technical Readiness Level (TRL) research applications into operational environments.

The Dynamic Weaver Framework is applied to the Advanced Diagnostic System, under development at Ames Research Center, which is composed of a set of Diagnostic Client Applications developed from multiple funding sources. While DCAs (agents) have utility as stand-alone applications, they generate knowledge about the system that can be shared throughout the ADS, thereby increasing the overall effectiveness of the ADS as a diagnostic system. To facilitate knowledge sharing and reuse in the ADS, we present a basic ontology that defines the vocabulary by which agents can exchange knowledge within the ADS universe.

Using the Dynamic Weaver Framework to support code reuse and simplify reconfiguration and the basic ADS ontology to enhance the effectiveness of the ADS enables the systems integrator to bring a better software system into the operational environment at a lower cost.

[13] In the ADS architecture, the <Host> and <Port> used for DCA data product subscription information belong to the Diagnostic Data Server. A DCA publishes only to the DDS; the DDS then publishes to all of the subscribers, relieving a DCA of the burden of keeping track of all its subscribers.

References

[1] D. Bardou, "Roles, Subjects, and Aspects. How Do They Relate?," Position paper, *ECOOP '98 Workshop on Aspect-Oriented Programming,* July 20–24, 1998.

[2] L. Berger, M. Dery and M. Fornarino, "Interactions Between Objects: An Aspect of Object-Oriented Languages," Position paper, *ECOOP '98 Workshop on Aspect-Oriented Programming,* July 20–24, 1998.

[3] M. Yuan and N. Richards, "Lightweight Aspect-Oriented Programming," *Dr. Dobb's Journal 351, 18–22,* August 2003.

[4] T. Barrett, "Dynamic Proxies in Java and .NET, Separating cross-cutting concerns," *Dr. Dobb's Journal 350, 18–26,* July 2003.

[5] C. Lopes and G. Kiczales, "Recent Developments in AspectJ," *ECOOP '98 Workshop on Aspect-Oriented Programming,* July 20–24, 1998.

[6] J. Pryor and N. Bastán, "A Reflective Architecture for the Support of Aspect-Oriented Programming in Smalltalk," Position paper, *ECOOP '98 Workshop on Aspect-Oriented Programming,* July 20–24, 1998.

[7] B. Bershad, S. Savage, P. Pardyak, G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chamber, "Extensibility, Safety and performance in the SPIN Operating System," *15th Symposium on Operating System Principles*, December 3–6, 1995.

[8] J.A. Dickerson and B. Kosko, "Virtual Worlds as Fuzzy Cognitive Maps", IEEE 1993.

[9] Bart Kosko, *"Neural Networks and Fuzzy Systems"*, Prentice Hall, Inc., 1992.

[10] T. Gruber, "What is an Ontology?," http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[11] W3C Candidate Recommendation, "OWL Web Ontology Overview," http://www.w3.org/TR/owl-features/ , August 18th, 2003. Copyright © 2003 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231.

[12] David Perkins and Evan McGinnis, *Understanding SNMP MIBs,* Prentice Hall PTR, 1997.

[13] D. Fletcher, R. Alena, "A Scalable, Out-of-Band Diagnostics Architecture for International Space Station Systems Support", *2003 IEEE Aerospace Conference Proceedings,* March 8–15, 2003.

[14] C.D. Stylios, P.P. Groumpos, "Fuzzy Cognitive Map in Modeling Supervisory Control Systems", *Journal of Intelligent and Fuzzy Systems*, Vol. 8, No. 2, pp. 83-98, 2000.

[15] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufmann Publishers, Inc., 1988.

[16] R. Sterritt, A.H. Marshall, C.M. Shapcott, S.I. McClean, "Exploring Dynamic Bayesian Belief Networks for Intelligent Fault Management Systems", *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, V, pp. 3646-3652, Sept. 2000.

BIOGRAPHY

Daryl Fletcher *received his B.S. degree in Applied Mathematics in 1993 and an M.S. degree in Engineering in 1995 from the University of Colorado-Boulder and is currently a Ph.D. student in Computer Science at the University of Colorado-Denver. His background is in systems development for aviation meteorology and development of Network Management Systems for data and voice networks. His research interests include applications of computational intelligence for the modeling, diagnosis and prognosis of complex systems. Prior to joining SAIC, he was a software developer at the National Center for Atmospheric Research in Boulder, CO. and a consultant to Lucent Technologies and Level(3) Communications, Inc.*

Faisal Akkawi*, whose area is software architecture for concurrent systems, has been on the faculty of Illinois Institute of Technology from 1998 to 2002. Currently he is an adjunct Faculty in the Department of Computer Science at Northwestern University. His research interests include software architecture for concurrent software systems, reactive/adaptive intelligent systems and design issues of concurrent programming languages.*

Richard Alena *is a Computer Engineer and the Group Lead for the Intelligent Mobile Technologies (IMT) Lab and the Mobile Exploration System (MEX) testbed at NASA Ames Research Center. The IMT team integrates mobile hardware and software components into unique systems capable of extending human performance aboard spacecraft during flight and payload operations. He was principal investigator for the Wireless Network Experiment flown aboard Shuttle and Mir, technology later adopted by the International Space Station Program. Rick spent three summers in the Canadian Arctic developing mobile technologies for human planetary exploration. He has a MSEE&CS from University of California, Berkeley.*

Daniel Duncavage *joined the International Space Station team as a civil servant at Johnson Space Center after completing his BSME and MSME at Northeastern University in 1994. After almost five years working with the Russian Space Agency handling management issues concerning the US research work being performed on the Russian Mir space station, Mr. Duncavage moved to the ISS Avionics Office to improve onboard diagnostics. The first set of tools he brought to the Station were employed to save the Control Moment Gyros on ISS Flight 3A. This success lead to the expansion of the effort that evolved into the Advanced Diagnostic Systems R&D project, led by Mr. Duncavage.*